Low-Latency In Situ Image Analytics With FPGA-Based Quantized Convolutional Neural Network

Low-Latency *In Situ* Image Analytics With FPGA-Based Quantized Convolutional Neural Network

Maolin Wang, Kelvin C. M. Lee, Bob M. F. Chung, Sharatchandra Varma Bogaraju, Ho-Cheung Ng, Justin S. J. Wong, Ho Cheung Shum, Kevin K. Tsia, *Member, IEEE*, and Hayden Kwok-Hay So, *Senior Member, IEEE*

***Abstract*— Real-time *in situ* image analytics impose stringent latency requirements on intelligent neural network inference operations. While conventional software-based implementations on the graphic processing unit (GPU)-accelerated platforms are flexible and have achieved very high inference throughput, they are not suitable for latency-sensitive applications where real-time feedback is needed. Here, we demonstrate that high-performance reconfigurable computing platforms based on field-programmable gate array (FPGA) processing can successfully bridge the gap between low-level hardware processing and high-level intelligent image analytics algorithm deployment within a unified system. The proposed design performs inference operations on a stream of individual images as they are produced and has a deeply pipelined hardware design that allows all layers of a quantized convolutional neural network (QCNN) to compute concurrently with partial image inputs. Using the case of label-free classification of human peripheral blood mononuclear cell (PBMC) subtypes as a proof-of-concept illustration, our system achieves an ultralow classification latency of 34.2 μs with over 95% end-to-end accuracy by using a QCNN, while the cells are imaged at throughput exceeding 29 200 cells/s. Our QCNN design is modular and is readily adaptable to other QCNNs with different latency and resource requirements.**

***Index Terms*— Cell image classification, convolutional neural network (CNN), field-programmable gate array (FPGA), hardware architecture, low-latency inference, multiplexed asymmetric-detection time-stretch optical microscopy (multi-ATOM), quantized convolutional neural network (QCNN), reconfigurable computing.**

Manuscript received March 25, 2020; revised September 8, 2020; accepted December 8, 2020. This work was supported in part by the Croucher Foundation Croucher Innovation Award 2013; in part by the Innovation and Technology Commission under Grant ITS/204/18; in part by the Research Grants Council of Hong Kong under Grant CRF C7047-16G and Grant CRF 17307919, Grant GRF 17208918, Grant GRF 17209017, and Grant GRF 17245716; in part by the University of Hong Kong Platform Technology Fund; and in part by the National Natural Science Foundation of China (NSFC) under Scheme Excellent Young Scientists Fund (Hong Kong and Macau) (Project Number: 21922816). *(Corresponding author: Hayden Kwok-Hay So.)*

Maolin Wang, Kelvin C. M. Lee, Kevin K. Tsia, and Hayden Kwok-Hay So are with the Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong (e-mail: hso@eee.hku.hk).

Bob M. F. Chung and Ho Cheung Shum are with the Department of Mechanical Engineering, The University of Hong Kong, Hong Kong.

Sharatchandra Varma Bogaraju is with the Faculty of Computing, Engineering and the Built Environment, Ulster University, Jordanstown Campus, Newtownabbey BT37 0QB, U.K.

Ho-Cheung Ng is with the Department of Computing, Imperial College London, London SW7 2AZ, U.K.

Justin S. J. Wong is with Conzeb Ltd., Hong Kong.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2020.3046452.

Digital Object Identifier 10.1109/TNNLS.2020.3046452

I. Introduction

RECENT advances in the deep neural network have given image analytics powerful tools to extract the unprecedented amount of information by learning from massive data sets. For the off-line, batch mode analysis, a number of general-purpose [1], [2] and domain-specific [3], [4] software frameworks are readily available for both training and inference. Using state-of-the-art accelerators, such as graphic processing units (GPUs), these software frameworks have enabled researchers to study and develop even the most complex application-specific neural networks with relative ease with high processing throughput.

However, for online applications that require *in situ* single item image analytics for immediate feedback control, real-time implementation of such complex neural network inference operation remains an open challenge. Existing acceleration techniques using parallel computers or GPUs, while being able to improve batch processing throughout, are not designed for real-time applications that have the additional requirements on low processing latency and guaranteed performance. In autonomous vehicle platooning control [5], for instance, complex control decisions must be made by analyzing input, such as the array of multispectral cameras and LIDAR with low-processing latency. Similarly, in many novel cell-based applications that are instrumental in cell biology research and drug discovery, such as in high-speed cell sorting and encapsulation, real-time actuation of each individual cell after it is being imaged is required [6]–[8]. In these application scenarios, the image analytic latency, as measured from the time when an image is produced to the time when an analytic decision can be made about that object, must be kept as low and predictable as possible to minimize its impact on the overall system design. Consider, for instance, the case of actuating cells that are imaged while flowing in a microfluidic channel at speed of 1 m/s, such as those demonstrated in [9] and [10]. Every d additional microseconds of decision latency will lead to a corresponding increase of d micrometers in the microfluidic channel that requires precise control. In the system developed in [11], the proposed GPU accelerated cell classifier using a deep convolutional neural network (CNN) incurred 3.2 ms of inference latency, which translates into a 3.2 cm microfluidic channel that the authors had to tightly control for cell sorting at the end. While the authors have

developed a hardware-based cell velocity estimator to control its sorting mechanism, it imposed substantial overhead to the already complex system design.

Here, in this work, instead of relying on GPU for acceleration, we demonstrate that a hardware-only quantized CNN (QCNN) implementation is superior in providing predictable and ultralow inference latency suitable for a range of demanding real-time applications. We show that a field-programmable gate array (FPGA)-based reconfigurable computing system is superior in these application scenarios as it can efficiently serve both as a low-level hardware integration platform and as a high-level computing device for complex analytics computations.

To illustrate this integrated hardware-processing concept, we adopted this system architecture to a recently developed quantitative label-free imaging flow cytometry technique, coined multiplexed asymmetric-detection time-stretch optical microscopy (multi-ATOM), that achieved a high imaging line-scan rate of 11.8 MHz [9]. The entire image processing chain, from interfacing to the laser imaging sources and controlling the two data acquisition analog-to-digital converters (ADCs), to image formation and cell detection and classification using a QCNN, are performed in hardware within the central FPGA processing unit (CFPU). This unified, deeply pipelined process chain allows the QCNN inference operation to execute in parallel to the series of cell image formation operations, thus ensuring that the lowest possible overall classification latency by the time a cell is completely imaged.

The proposed QCNN featured a layer-parallel architecture that was designed to perform inference on a single input image without batch processing. The hardware design of each layer, including the convolutional layer, the pooling layer, and the fully connected layers, were parameterizable to facilitate resource-latency tradeoff. Furthermore, computations were performed with an 8-bit number representation with quantized weights, which matches our input image data and provided resource-efficient hardware implementations. Finally, our hardware QCNN design was deeply pipelined to maintain high processing throughput that was compatible with the input cell imaging system.

With this system, we demonstrated an ultralow cell classification latency of 34.2 μs, i.e., three orders of magnitude reduction from state-of-the-art GPU-accelerated software designs, with over 95% end-to-end accuracy when classifying the subtypes of human peripheral blood mononuclear cells (PBMCs) at a real-time throughput of 29200 cells/s. In addition, compared with software-based solutions, our FPGA-based analytic system is capable of producing results with fixed and predictable latency that is needed for precise feedback control.

This work, thus, advances the state of the art in the following areas.

1) We demonstrate a first-of-its-kind hardware-only QCNN inference machine co-optimized with the real-time imaging front end to perform real-time *in situ* cell classification with low and predictable latency.
2) We propose a streaming architecture for mix-granularity pipelined operations, from image formation to cell classification, which allows flexible latency–area tradeoff while maintaining high data processing throughput.
3) Using the hardware-based QCNN inference implementation, we demonstrate an ultralow-latency *in situ* image-based cell classifier with 34.2-μs processing latency and 95% end-to-end accuracy.

In Section II, a brief review of related work in hardware-based neural network inference will first be presented. An overview of the *in situ* cell image analytic platform and the integration between image formation and neural network inference will be shown in Section III. Details about our QCNN implementation will be shown in Section IV. Performance evaluation of the system will be shown in Section V before we conclude this article in Section VI.

II. Related Works

Efficient hardware implementations of deep neural network inference have been a topic of intense research interest. Compared with typical software-based GPU-accelerated solutions, hardware implementations of neural network inference have been demonstrated with superior power-efficiency, making them particularly useful for power-limited systems, such as autonomous vehicles and intelligent edge devices. The use of application-specific architectures, both from a system and microarchitecture's point of view, and custom numeric precision computations have both been keys in enabling such achievements.

In one of the earliest work, Farabet *et al.* [12] presented a custom architecture for performing CNN inference on FPGA. Subsequently, a number of large-scale frameworks have been developed [13]–[17], each based on their own custom architecture for accelerating neural network inference, but with a common thread focusing on developing efficient matrix computations for maximum processing throughput.

Another important optimization strategy for hardware inference is to utilize nonstandard or reduced precision arithmetic for inference. As observed in [18], the redundancy presented in the vast parameter pool of parameters in deep neural networks allows inference to compute with reduced-precision arithmetic operations without significant effect on the model accuracy. This allows researchers to utilize fixed-point or integer arithmetic in place of more hardware-demanding floating-point operations for most of the demanding operations. To that end, a large body of works [19]–[21] has already demonstrated neural network inference implementations with 8-bit weights on FPGAs with negligible accuracy drop. Jacob *et al.* [22] presented a series of models using 8-bit weights and activation to provide a tradeoff between computation latency and classification accuracy. Tripathi *et al.* [23] also demonstrated a CNN with 8-bit weights for object detection while maintaining the model accuracy to be as good as its floating-point counterpart. Baskin *et al.* [24] implemented AlexNet inference on FPGA using 2-bit activation and demonstrated energy efficiency over GPU. Pushing the technology limitation further, a number of works have also studied the use of extreme low-bit-width arithmetic, such as binary [25]–[30] and ternary weights [31]–[33] in various neural network inference operations, while maintaining acceptable model performance degradation.

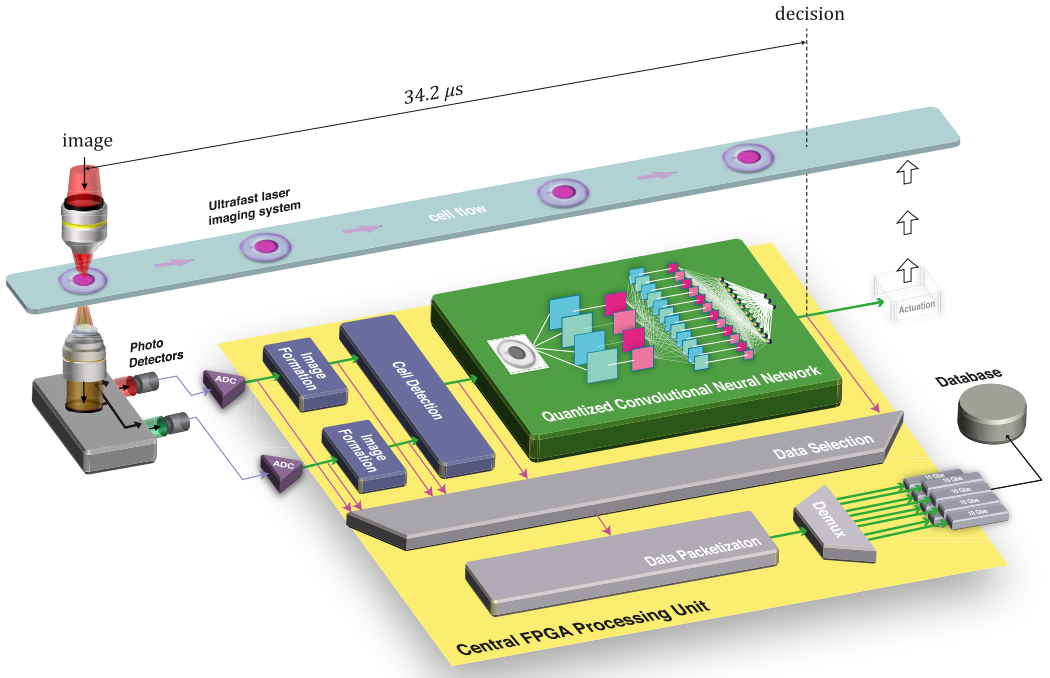


Fig. 1. FPGA-based reconfigurable computing platform for intelligent *in situ* image analytics in real time. The CFPU serves as a unified platform for concurrent image formation, cell detection, and classification using a QCNN. Fully pipelined stream processing FPGA gateware results in predictable and ultralow latency at high throughput.

While the above QCNN designs on FPGA have successfully resulted in good throughput performance, they were not particularly optimized for low-latency operations. To address the need for low-latency inference, Venieris and Bouganis [34] proposed a latency-driven design methodology for mapping QCNN on FPGA. A flexible architecture was designed that can be automatically derived based on the QCNN workload and FPGA resources using a synchronous dataflow computation model. Ma *et al.* [35] have implemented quantized VGG-16 on FPGA for ImageNet classification. By analyzing the loop operation and dataflow in FPGA, a latency of 48 ms can be achieved by balancing computation and memory traffic. Finally, Geng *et al.* [36] have designed a low-latency binarized neural network inference of AlexNet, VGGNet, and ResNet. Our design also aimed at reducing inference latency but with an additional requirement of maintaining a deterministic latency. Furthermore, we achieved these goals by employing a customized hardware architecture and by using 8-bit quantized operations that matched with our imaging front end. Finally, our QCNN was tightly coupled with the imaging formation hardware to allow true *in situ* inference on cell images concurrently as they were formed. To the best of our knowledge, this is the first work that addresses total end-to-end inference latency with system-level co-optimization between the imaging front end and the hardware neural network inference engine to achieve an ultralow classification latency of 34.2 μs.

III. System Architecture

Fig. 1 shows an overview of our *in situ* cell image analytic platform. Our target QCNN was implemented inside the central FPGA that provided a unified reconfigurable computing platform for both low-level image formation and high-level image analytics. This tight integration allowed us to co-optimize the image formation and analytics operations and to employ an end-to-end stream-based processing model across all tasks that maximized concurrence through careful pipelining at multiple levels of granularity. Most importantly, such tight integration allowed our QCNN classifier to perform image classification *in situ*. The QCNN inference computation commenced as soon as the first few lines of input cell image was produced, and it continued to operate on the subsequent image in parallel as soon as they were produced.

In the following, the coarse-grained system-level pipeline that included the optical front end, the image formation process, data filtering, and, finally, QCNN inference will first be described. The details of our fine-grained QCNN pipeline will be shown in Section IV.

A. Concurrent Image Formation and Image Analytics

At the system level, coarse-grain parallelism in a unified platform allowed our system to perform all subtasks of image formation and image analytics concurrently as a pipeline. This allowed analytic operations to commence before the input images were completely formed. [see Fig. 2(a)].

In our current implementation with the multi-ATOM imaging front end [9], images of the fast-flowing target cells in a microfluidic channel (at speed exceeding 1 m/s) were captured by ultrafast laser line-scan illumination that was orthogonal to the cell flow. The distinctive feature of multi-ATOM is its ability to generate multiple raw image (phase-gradient) contrasts of the same cell in each line scan. It has been demonstrated that these multiple (four in total in this work)

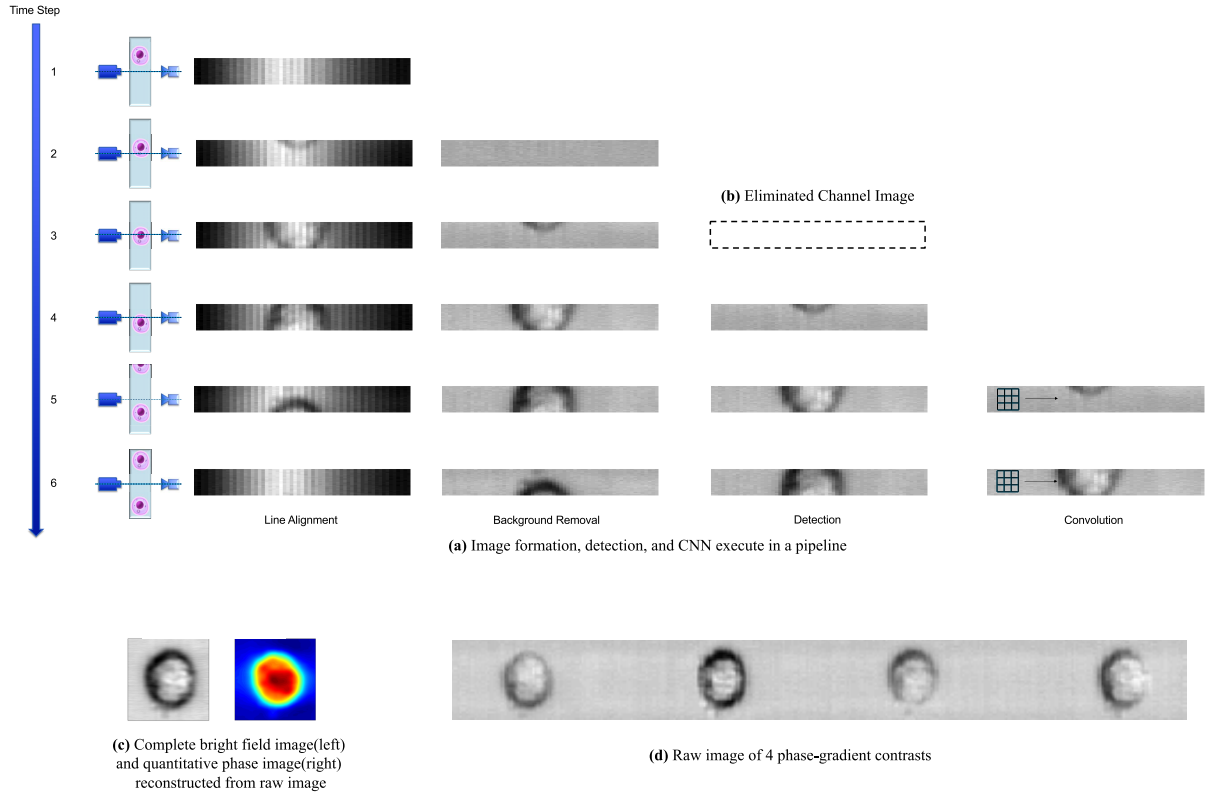


Fig. 2. Concurrent image formation and analytics in a pipeline (using ultrafast flowing cell imaging by multi-ATOM as an example). (a) Snapshots of the image formation process as a pipeline spanning multiple time steps (image formation, detection, and CNN execute in a pipeline). Every step operates on a partial image such that analysis of an image may commence before the rest of the image is formed (last row). Note that only one raw image contrast is shown here for simplicity. (d) In practice, four different image phase-gradient contrasts of the same cell are acquired in multi-ATOM. (b) Cell detection step selects only relevant cell image to be processed by CNN (eliminated channel image). (c) Complete bright field image (left) and QPI (right) reconstructed from raw image. (d) Raw image of four phase-gradient contrasts.

contrasts can be harnessed to compute quantitative phase images (QPIs), which quantifies a multitude of valuable physical and mechanical cellular properties indicative of cell states and functions [37], [38].

At each time step i [moving downward in Fig. 2(a)], each subtask j of the pipeline (including image alignment, background removal, cell detection, and the first convolution layer of our CNN) passed the processing result from time $i-1$ to the right, while it began processing a new partial image data delivered from task $j-1$ on the left. Because of this pipeline operation, as illustrated on the last two rows of the figure, the sliding window operation of first convolution layer conv1 (rightmost column) was able to commence on the first part of a detected cell, while the rest of the cell was still being imaged (leftmost column). Although not shown in Fig. 2(a), this pipeline pattern continued throughout our CNN classifier until the final classification decision was formed (see Section IV for details).

It is worth noting that, because of this pipeline operation, the complete image of a detected cell [see Fig. 2(c)] was never available as a whole at any point in the system during real-time *in situ* analytics. In applications where the complete recording of the imaging channel or the detected cell images are needed, they can be selected through the on-chip selection network and transmitted to the data aggregation cluster for further analysis. Furthermore, while each image snapshot shown in Fig. 2(a) contains 25 lines of the input for illustration purposes, in practice, our hardware implementation (see Section III-B) operated on a much finer granularity of 16-pixel blocks that were produced by the sampling ADC.

The aforementioned unification of image formation and analytics into one seamless pipeline allows co-optimizations between the two that are otherwise difficult to achieve. To reduce complexity in the current imaging front end, the usual step of QPI reconstruction in the original multi-ATOM [see Fig. 2(c)] was omitted. Instead, our classifier took the four raw phase-gradient images of each cell and stitched them together into a single image [see Fig. 2(d)] for training and inference by the QCNN. The integrated pipeline also allowed us to adopt for the limited ADC bandwidth by compensating the reduced cell image resolutions with QCNN implementations that achieved similar accuracy at the same data throughput.

B. Fine-Grained Reconfigurable Image Processing Hardware Pipeline

In addition to the coarse-grain parallelism at the system level, implementing our image analytics system on FPGA further allowed us to exploit fine-grain parallelism at the gateware implementation level. It was particularly important for the stages closest to the imaging source where the data bandwidth requirement was stringent.

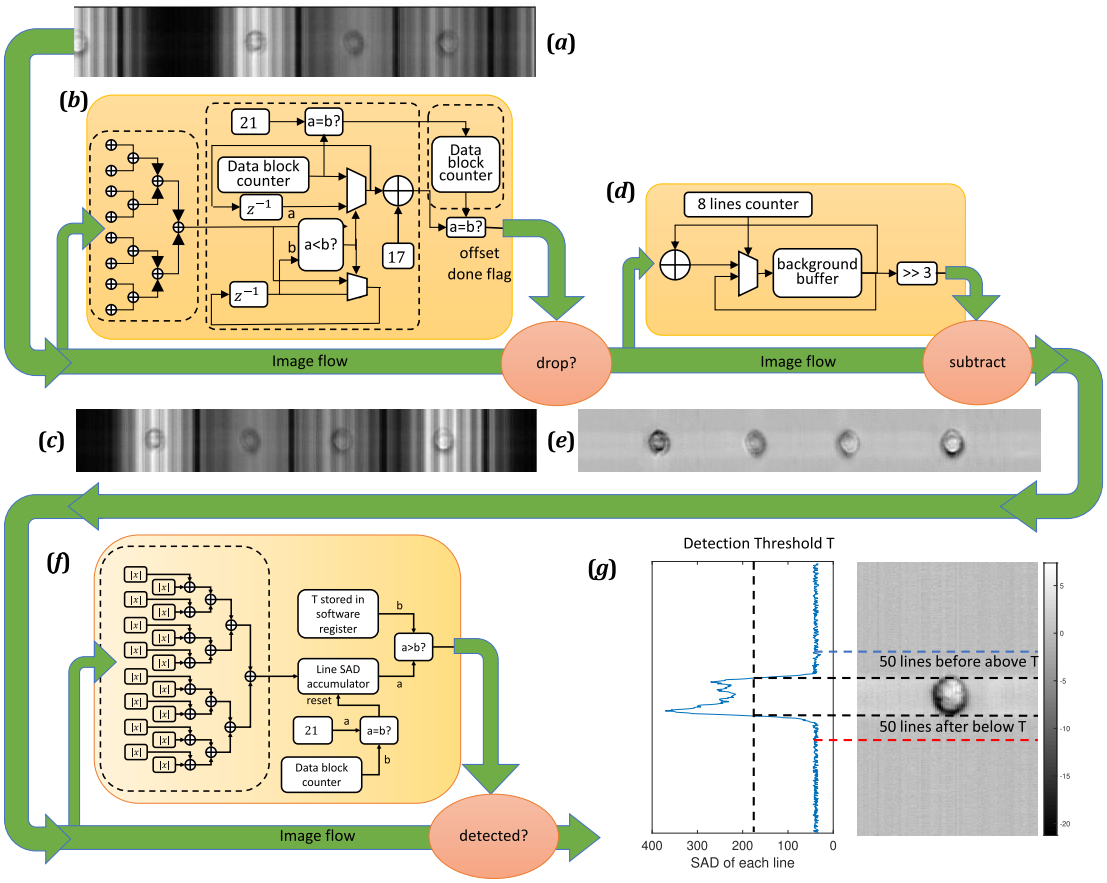


Fig. 3. Processing pipeline design. (a) Raw image before line alignment. (b) Line alignment hardware. (c) Image after line alignment. Note the four phase-gradient contrasts of the cell samples are now centered. (d) Background computation and removal hardware. (e) Image after background removal. (f) Object detection hardware. (g) SAD values of lines around a target cell. Fifty lines before and after SAD exceeding a threshold T are included as part of the detected object.

To illustrate this need for fine-grained pipeline design, consider our current application that interfaced with the imaging front end through a pair of high-speed ADCs. On each channel, the ADC produced 8-bit samples at 3.9648 GHz and transmitted 16 data in parallel to the FPGA at the system clock rate of 247.8 MHz, which resulted in a sustained data input throughput of 3.9648 GB/s. In other words, to process these ADC samples in time, the first few stages of the image processing pipeline must be able to process 16 pixels in parallel every cycle.

Consequently, both the line alignment module [see Fig. 3(b)] and the background removal module [see Fig. 3(d)] employed a similar design strategy that relied on an off-datapath module for estimation, while the main datapath maintained the input data rate.

In the case of line alignment, the estimator observed the first usable line of input to determine an offset that allowed the system to center the cells in the images. It was achieved by summing the values of each of the 16-pixel blocks with an adder tree as they were read from the ADC. Each image line contained 21 blocks, and the index of the block with the lowest summed value was determined, which corresponds to the dark color band in Fig. 3(a). Subsequent lines of the cell images were aligned with respect to this index. The main datapath dropped all pixels until the correct alignment was obtained, and it maintained the input data throughput from then on. Details of the line alignment operation are shown in Algorithm 1.

Similarly, the background removal subsystem also relied on an estimator that determined the background image by analyzing the incoming image lines in parallel to the main dataflow. Two common statistical methods (mean and median) were considered for estimating background values along with the image column data. The median method was used by our collaborators for the off-line analysis since it was more robust against sparse and asymmetric variations of pixel values in the raw data and was able to provide more accurate background estimations over the mean method in some cases. However, the median computation involved data sorting, which introduced substantially higher latency than the simpler hardware pipeline for mean computation. Therefore, to maximize the latency headroom for the remaining image analysis stages, the mean method was adopted. Pixel values were accumulated across eight rows of image data and then divided by 8 using simple right shift by 3 bits [see Fig. 3(d)] to obtain the mean values as background estimation along the image column of 336 pixels. The resultant background values were then subtracted from the original image data as they streamed through

Algorithm 1 Line Alignment

```
/* Each scan line in our input
   contained 21 × 16 = 336 pixels                    */
Input : 16-pixel block stream b_j^(i),
        line scan number i ∈ {1, 2, ···},
        block index j ∈ {1, 2, ..., 21},
Output: Aligned 16-pixel block stream B_j^(i)
/* First line after initialization                   */
(p, minBlkVal) ← (1, ∞)
for j ← 1 to 21 do
    s_j ← sum of 16 pixel values within b_j^(1)
    if s_j < minBlkVal then
        (p, minBlkVal) ← (j, s_j)
    end
end
/* Shift remaining lines by p blocks                 */
for i ≥ 2 do
    (B_1^(i−1), ..., B_j^(i−1), ..., B_21^(i−1)) ←
    (b_p^(i), ..., b_21^(i), b_1^(i+1), ..., b_{p−1}^(i+1))
end
```

Algorithm 2 Background Removal

```
Input : Aligned block stream B_j^(i),
        line scan number i ∈ {1, 2, ···},
        block index j ∈ {1, 2, ..., 21}
Output: Background-free 16-pixel block stream C_j^(i)
for j ← 1 to 21 do
    background accumulation buffer T_j ← 0
end
/* Estimate background from the average
   value of the first 8 lines                        */
for i ← 1 to 8 do
    for j ← 1 to 21 do
        T_j ← T_j + B_j^(i)
    end
end
for j ← 1 to 21 do
    T_j ← T_j ≫ 3                    // divide by 8
end
/* All remaining lines                               */
for i ≥ 9 do
    for j ← 1 to 21 do
        C_j^(i−8) ← B_j^(i) − T_j
    end
end
```

the system to obtain pure cell images [see Fig. 3(e)] for the next processing stage. Algorithm 2 describes this background removal operation in more detail.

The cell detection module marked the point where the stringent dataflow requirement began to relax as it filtered out image data that would unlikely be needed for further analysis [see Fig. 2(b)]. To facilitate efficient fine-grain pipeline implementation at high bandwidth, a hardware-optimized stream-based cell detection scheme was developed. The proposed detection modules employed a thresholding scheme based on the sum of absolute difference (SAD) of a scan line against the moving average [see Fig. 3(g)].

IV. QCNN Implementation on FPGA

Classifying cell images with CNN was the most computationally demanding subsystem in our processing pipeline. It was the major contributor to the overall classification latency and limited system hardware processing throughput. The main challenge of our *in situ* QCNN design rested on the need to maintain the high stream-processing throughput as determined by the imaging front end and input ADC sampling speed while achieving ultralow latency and high-accuracy classification under tight hardware resource constraints. Our design addressed these challenges by adopting a fully pipelined, parallel design of the CNN that operated on image stream immediately without buffering the whole image. Furthermore, our neural network operated with reduced-precision arithmetic as a QCNN, which reduced hardware resource requirements and computation latency at the same time. Finally, based on this CNN architecture, a system-level design space exploration was performed to tradeoff between latency, throughput, and classification accuracy under resource constraints.

A. Hardware Architecture

Our CNN consisted of two pairs of convolutional-maxpooling layers followed by two fully connected layers [see Fig. 4(b)].

Input to the CNN was 336×336 pixel cell images generated from the cell detection module, while the CNN produced classification results as output. As a continuation of the stream processing paradigm for image formation, all the layers were designed to operate in parallel as a pipeline [see Fig. 4(a)]. The layers, thus, relied only on partial results from its previous layer or the image formation subsystem for streaming computation.

The network layout and the choice of metaparameters, from the sizes and strides amount of the kernels to the number of layers employed, were designed specifically to facilitate efficient hardware implementation. For instance, in the input convolution layer (conv1), the kernel size was chosen to be 8×8 pixels—a relatively large square shape with a power of two number of pixels on the side—to allow efficiently balanced adder trees to be implemented in hardware. A large stride value of 4 was also chosen as a means to reduce the amount of hardware resource consumption. These architectural choices allowed efficient 2-D convolution operations to be carried out by a pair of streaming convolutional units (SCUs).

Inspired by the idea proposed in [39] that optimized computation efficiency, our SCU was optimized for reduced computation latency by performing the partial sum of all kernels[1] that covered the 16 input pixels every cycle in parallel [see Fig. 4(c)]. Finally, four such hardware structures were instantiated, one for each feature kernel for the conv1 layer [see Fig. 4(a)].

[1]The exception being that the last pair of kernels were delayed for computation in the following cycle when following 16 pixels were input.

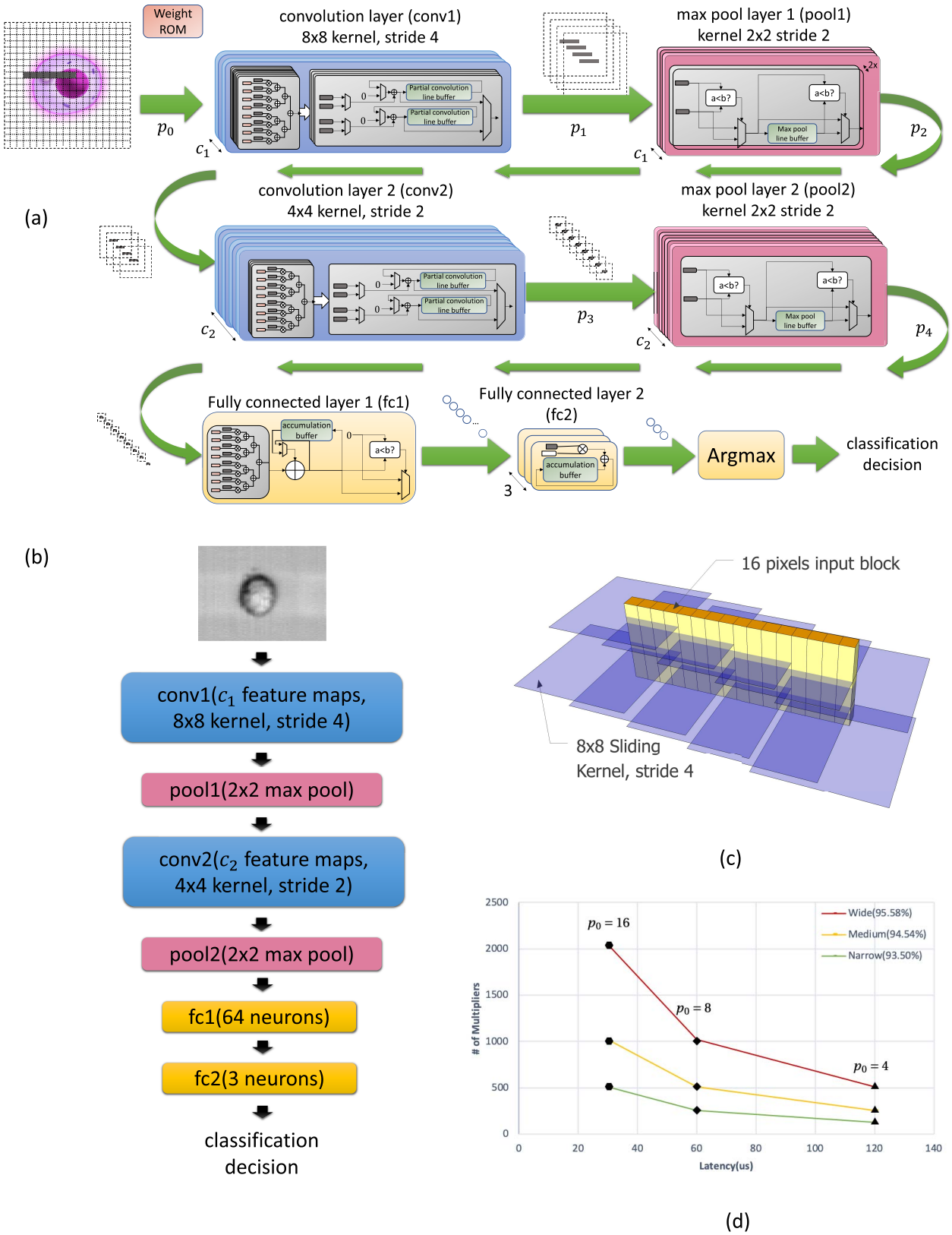


Fig. 4. QCNN design. (a) Hardware architecture overview. A fully pipelined parallel implementation of QCNN where all layers operate concurrently. The architecture accepts 16-byte data block every cycle. (b) Layer configuration of the proposed QCNN. (c) Partial products of all sliding kernels that intersect with the current input data block are computed concurrently in a single cycle. Kernels from conv1 shown for illustration. (d) Latency and hardware resource usage analysis of the proposed QCNN.

This streaming processing model continued through the interference operations with careful data rate control until the end of the final fully connected layer (fc2). Such a streaming process not only reduced the amount of on-chip memory required for buffering but also significantly reduced our classification latency.

B. Continuous Inference

From a data analytic perspective, our streaming architecture was designed to perform inference operations on each individual cell as it was imaged without first being buffered to form a data processing batch. Compared with conventional throughput-optimized CNN implementations that often operated with input data batches, single datum inference operation avoided the need for batch buffering and was, thus, particularly suitable for our latency-optimized designs. Specifically, let N_b be the batch size and t_{img} be the time required to form a cell image. The first image needed to wait for the formation of all N_b images in the batch before being sent to GPU for computation. Similarly, the second image is needed to wait for $(N_b - 1)t_{\text{img}}$, while the last image is still needed to wait for the formation of itself. Therefore, the average additional waiting time for each image due to batch buffering was

$$\frac{N_b + (N_b - 1) + (N_b - 2) \cdots + 2 + 1}{N_b} t_{\text{img}} = \frac{1}{2}(N_b + 1)t_{\text{img}}.$$

C. Quantization

While the streaming architecture described above promised to achieve ultralow classification latency at high throughput, it required a significant amount of configurable hardware resources of the FPGA to physically implement operations from all layers in parallel. As a result, the size of the CNN implementable was ultimately constrained by the number of available resources on the FPGA.

To reduce the number of configurable hardware resources required, a QCNN implementation was adopted [22], [25], [40]. In our implementation, all weights and activation functions were represented as 8-bit fixed-point numbers during inference. We used the network quantization method proposed in [22] to explore efficient bitwidth of weight and activation. Floating-point weights were used to accumulate the gradient update during backward pass. During forward pass, weights and activations were quantized using integer quantization training scheme proposed in [22]. As illustrated in Table I, the classification accuracy of our CNN naturally decreased as the number of bits used to quantize the neuron weights decreased. Yet, compared with the original single-precision floating-point implementation in software, the accuracy of our QCNN remained competitive even at 8-bit quantization. The network training only failed to converge when the weights were quantized to fewer than 5 bits. As a result, to facilitate a simple hardware design that matched with the 8-bit pixel input from the ADC, an 8-bit number representation was chosen for our current QCNN implementation.

TABLE I
MODEL DESIGN EXPLORATION

Model	Accuracy[1]	Feature maps conv1	Feature maps conv2	# of parameters
Narrow(fp32)	93.50%	4	8	26,080
Medium(fp32)	94.54%	8	16	85,787
Wide(fp32)	95.58%	16	32	341,555
Narrow(int8)	93.62%	4	8	26,080
Narrow(int7)	93.27%	4	8	26,080
Narrow(int6)	93.15%	4	8	26,080
Narrow(int5)	90.18%	4	8	26,080

[1] Average validation accuracy of 5 fold cross validation

D. Design Space Exploration

In order to design a neural network model that achieved low inference latency under the constraints set by: 1) the high input data rate imposed by the imaging front end and 2) limited reconfigurable resource and I/O bandwidth of the FPGA system, a design space exploration was conducted.

During our exploration, three models with a different number of feature maps, which directly affects their classification accuracy and resource implications, were considered, as shown in Table I. To avoid overfitting, we stopped increasing the width of the network when the training error of the widest model in Table I reached 99.94%.

Based on these network models, hardware implementations with different resource requirements subject to the image processing throughput constraint were explored. As shown in Fig. 4(a), hardware resources can be adjusted by varying the amount of parallelization employed in the QCNN implementation at the expense of reduced pixel processing rate.

Denote pixel processing rate, p_i, as the number of pixels passed as input to the $(i + 1)$th layer of our layer-parallel QCNN classifier per cycle. The activation output rate (p_i) of each pipeline stage i was balanced to avoid pipeline congestion. Let c_i, H_i, and W_i be the number, the height, and the width of stage i's output feature maps. It took $((c_{i-1}H_{i-1}W_{i-1})/(p_{i-1}))$ to feed all the input feature maps into stage i. The pipeline congestion-free constraint implied that all the output feature maps were fed into stage $i+1$ within the same period after stage i's processing delay. Hence, p_{i-1} and p_i satisfied the following equation:

$$\frac{c_{i-1}H_{i-1}W_{i-1}}{p_{i-1}} = \frac{c_i H_i W_i}{p_i}. \tag{1}$$

Note that both H_i and W_i were reduced to $(1/s_i)$ of their original size H_{i-1}, W_{i-1} after passing through stage i. After rearranging both sides of Equation (1), we obtained Equation (2) to describe the relation between p_{i-1} and p_i

$$p_i = p_{i-1} \cdot \frac{c_i}{c_{i-1}} \cdot \frac{1}{s_i^2}. \tag{2}$$

Given a CNN model specification, the data rate of each pipeline stage could be computed recursively using Equation (2) and was all proportional to p_0, the input pixel rate to the QCNN. As a result, we used p_0 as a hardware parallelism indicator.

Based on the value of p_0, we can determine the number of reconfigurable hardware resources needed for a particular QCNN implementation. For example, the number of multipliers for weight activation multiplication of convolutional stage i was

$$c_{i-1}\, p_i c_i \left(\frac{k_i}{s_i}\right)^2 \tag{3}$$

where k_i and s_i are the kernel size and stride of the convolutional layer. Since p_i was proportional to p_0, the overall hardware resources usage would also be proportional to p_0.

At the same time, p_0 was also closely related to the classification latency. Assume that the image size was N_{img}, and it took (N_{img}/p_0) clock cycles to feed the entire image into the pipeline. The classification latency would, therefore, become (N_{img}/p_0) + PIPELINEDELAY, which was roughly inversely proportional to p_0. Fig. 4(d) shows the relation between the number of multipliers required in the QCNN and the classification latency of the models in Table I.

In our implementations, DSP blocks on the FPGA were used exclusively for weight activation multiplications. As a result, the number of DSP blocks required can be derived from Equation (3) directly, which was also proportional to p_0.

In terms of on-chip memory, weights and partial convolution accumulations accounted for most of their usage in our design. The on-chip memory used for weight storage was related to the total number of parameters in each layer. In the case of convolution layers, it was proportional to the neural network width quantifying by the number of output feature maps c_i of each stage i. As the data coming into each convolution stage followed the same line-by-line manner, a convolution could only be completed after k_i lines of input. Partial convolutions of the first line were calculated immediately upon data arrival. The resulting partial accumulations were stored in a first-in-first-out (FIFO) waiting to be merged with other partial accumulations that required the next $k_i - 1$ lines' data. Note that each line was associated with other (k_i/s_i) convolutions due to the convolution window overlapping. For each output feature map of convolution stage i, (k_i/s_i) FIFOs of size W_i were used for partial convolution accumulation. Total on-chip memory used in stage i for partial convolution accumulations could, therefore, be described using the following equation:

$$c_i \cdot W_i \cdot \frac{k_i}{s_i} \tag{4}$$

where W_i is the linewidth of the output feature map. The same as the weight storage, on-chip memory usage of partial convolution accumulations was also proportional to network width.

The DSP and BRAM usage models of our QCNN hardware can be obtained by adding the resource usage of each stage i. Note that c_i, k_i, s_i, and W_i were network configuration parameters that determined classification accuracy, and the BRAM resource model in Equation (4) only contained these parameters. A larger network gave better classification accuracy, while it consumed more BRAMs. Except network configuration parameters, DSP usage was also affected by input data rate p_0 through Equation (3) and Equation (2), which directly affected classification accuracy. For the same network configuration, shorter classification latency did not require more BRAMs, but the QCNN hardware consumed more DSPs. Experimental results in hardware resource consumption that corresponds to this analytical model are shown in Section V. From Table II, we can see that, for the same network, using a large input data rate p_0 increases the number of DSP usage, while the number of consumed BRAMs remains the same.

V. RESULTS AND EVALUATIONS

A. System Setup

Our system was designed around the FPGA (Xilinx Virtex-6 XC6VSX475T) housed inside the ROACH-2 platform from the CASPER collaboration [41]. Two external ADCs (e2v EV8AQ160, 5 Gs/s, 8-bit) housed inside the CASPER ADC1 × 5000-8 board were used to interface with the multi-ATOM photodetector output. Four CPU nodes are connected to the ROACH-2 through eight 10-Gb/s Ethernet connections where data were transmitted as custom UDP packets for recording and analysis. A custom software aggregated and demultiplexed data from the four data aggregation nodes for downstream processing. The MATLAB 2012b with ISE system generator 14.5 was used for block diagram design and FPGA configuration file generation.

The imaging front end of our system was based on the multi-ATOM imaging system [9]. In short, our multi-ATOM system imaged fast-flowing single cells in a microfluidic channel based on an ultrafast line-scanning approach, in which a series of optical line scans was formed by illuminating broadband time-stretched laser pulses (center wavelength = 1064 nm, bandwidth = 10 nm, and repetition rate = 11.8 MHz) onto a diffraction grating. Images were subsequently reconstructed by stacking a series of line scans, while the targets flowed in the microfluidic imaging channel (>1 m/s).

The repetition rate of the laser pulse from the multi-ATOM imaging front end was used to serve as the master clock source for all remaining hardware processing clock domains. The laser pulse was first converted to an electrical pulse by FPD 310 photodetector from MenloSystems. The converted electrical pulse was then fed into a custom stretch and reshape circuit to produce a square wave master clock signal, which was synchronized to the original laser pulse with the same 11.8-MHz frequency. A Valon 5009 Dual Frequency Synthesizer was used to multiply the frequency of the master square wave clock signal by 168. Using the resulting 1982.4-MHz clock output from the synthesizer, the ADC further doubled the frequency internally to form a 3964.8-MHz sampling clock. Each laser scan pulse from multi-ATOM was, thus, sampled exactly 336 times by the ADC to form one image line. An internal clock division circuit inside the ROACH2 divided the 3964.8-MHz sampling clock into a 247.8-MHz system clock for the remaining hardware processing pipeline.

B. Classification Latency

Fig. 5(a) shows a plot of execution time for image formation and for various layers of the CNN to classify one cell. The pipeline action, in which different parts of the design execute

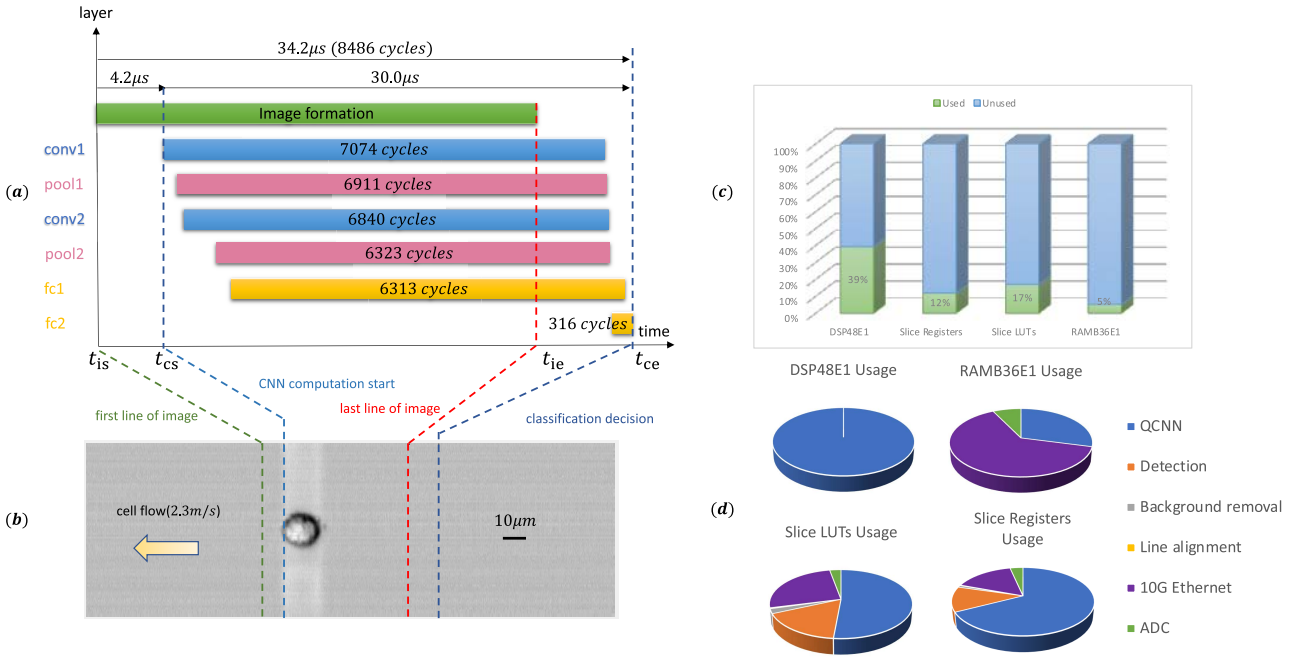


Fig. 5. Latency and resource consumption. (a) Plot showing the timing of image formation and various layers of CNN inference that execute in a pipeline. CNN computation begins as soon as a cell is detected. Depending on the exact timing when a cell is detected, the worst case latency for cell detection is $t_{ce} - t_{is}$. (b) Plot showing relative latency of CNN decision making compared with cell flow image. A classification decision is made soon after the sample cell is completely imaged ($t_{ce} - t_{ie}$). (c) Resource consumption of the design relative to available resources on target FPGA. (d) Resource consumption breakdown of each submodule.

concurrently, can easily be seen from the figure. Fig. 5(b) further maps the execution time to the corresponding cell image to highlight the spatial–temporal relationship between cell flow and classification latency.

To measure classification latency, recall that image lines of a cell were formed, while the cell flowed in the microfluidic imaging channel in our system. The cell detection module continuously analyzed the latest 50 lines of images to determine if a cell was presented. Once a cell was detected, the classification process using QCNN commenced. This inference operation continued as new image lines of the cell were formed. The CNN input ended after all 336 lines were captured, which translated into a complete 336×336 pixel input to the CNN. The inference operation is completed when all computations from the final fc2 layer are completed.

There are, thus, two important latency measurements. First, inference latency is the time measured from the start to the end of CNN inference operation ($t_{cs} \rightarrow t_{ce}$). Using an image size of 336×336 pixels, our CNN completed inference operations in 7436 cycles, which translated to 30.0 μs when the FPGA system was running at 247.8 MHz. Second, the detection latency must be taken into account, which measures from the start of an image to the time CNN operation starts ($t_{is} \rightarrow t_{cs}$). In the worst case, our cell detection module required 50 lines of input to detect a cell (4.2 μs) before it would trigger the start of CNN inference. As a result, the maximum classification latency our system is determined to be 34.2 μs ($t_{is} \rightarrow t_{ce}$).

Finally, combining Fig. 5(a) and (b) illustrates the spatial relationship between a cell flowing at 2.3 m/s and its corresponding classification result. It can be observed that, with our FPGA classifier, the classification decision is available at most 78.66 μm after a cell passes through the imaging channel.

C. Resource Consumption

Table II summarizes the detailed resources usage of each module in our hardware. We adopt the QCNN design configuration narrow and $p_0 = 16$ as wider QCNN clock frequency failed to catch up with the ADC sampling rate, which is closely coupled with the imaging front end, as explained in Section V-A. Fig. 5(c) shows the overall resource consumption of chosen classifier and other system components. Currently, the overall design consumes 39% of on-chip digital signal processing (DSP) units, 12% of slice registers, 17% of slice lookup tables (LUTs), and 5% of on-chip memory, which includes the QCNN implementation, the image formation hardware, ADC controllers, the data selection network, and other system support designs. Fig. 5(d) further breaks down the resource consumption according to different submodules. Among them, the QCNN, being the most computationally demanding, is responsible for almost all DSP block usages, while the eight 10-Gbe controllers consume the most on-chip memory for buffering.

D. Object Detection Accuracy

To evaluate the detection performance of our hardware image-based detection module, a continuous full system capture of both the phase-gradient contrast and fluorescence detection channels was performed on a flow of stained beads. From the full recording of the flow, by correlating against

TABLE II
SUMMARY OF FPGA RESOURCE USAGE

Usage of Each Module[1]	Slice Registers	Slice LUTs	Occupied Slicse	RAMB36E1	DSP48E1
Detection	8,419 (1.41%)	8,488 (2.85%)	3,028 (4.07%)	0	0
Background removal	625 (0.11%)	1,131 (0.38%)	342 (0.46%)	0	0
Line alignment	187 (0.03%)	215 (0.07%)	72 (0.10%)	0	0
10G Ethernet	11,236 (1.89%)	13,069 (4.39%)	4,375 (5.88%)	35 (3.29%)	0
ADC	2,412 (0.41%)	1,458 (0.49%)	558 (0.75%)	4 (0.38%)	0
QCNN narrow $p_0 = 4$	17,292 (2.91%)	10,340(3.47%)	4,561 (6.13%)	16 (1.50%)	214 (10.61%)
QCNN narrow $p_0 = 8$	26,756 (4.49%)	16,398 (5.51%)	6,626 (8.91%)	16 (1.50%)	406 (20.13%)
QCNN narrow $p_0 = 16$	47,309 (7.95%)	29,972 (10.07%)	9,972 (13.40%)	16 (1.50%)	790(39.18%)
QCNN medium $p_0 = 4$	29,515 (4.96%)	18,015 (6.05%)	7,305 (9.82%)	53 (4.93%)	406 (20.13%)
QCNN medium $p_0 = 8$	48,065 (8.08%)	28,488 (9.57%)	11,686 (15.71%)	53 (4.93%)	790 (39.18%)
Complete System[2]	70,707 (11.88%)	50,200 (16.87%)	21,444 (28.82%)	55 (5.17%)	790 (39.18%)
Available Resources	595,200	297,600	74,400	1,064	2,016

[1] The number in the parentheses are percentage usage.
[2] QCNN configuration choice in the complete system is narrow $p_0 = 16$.

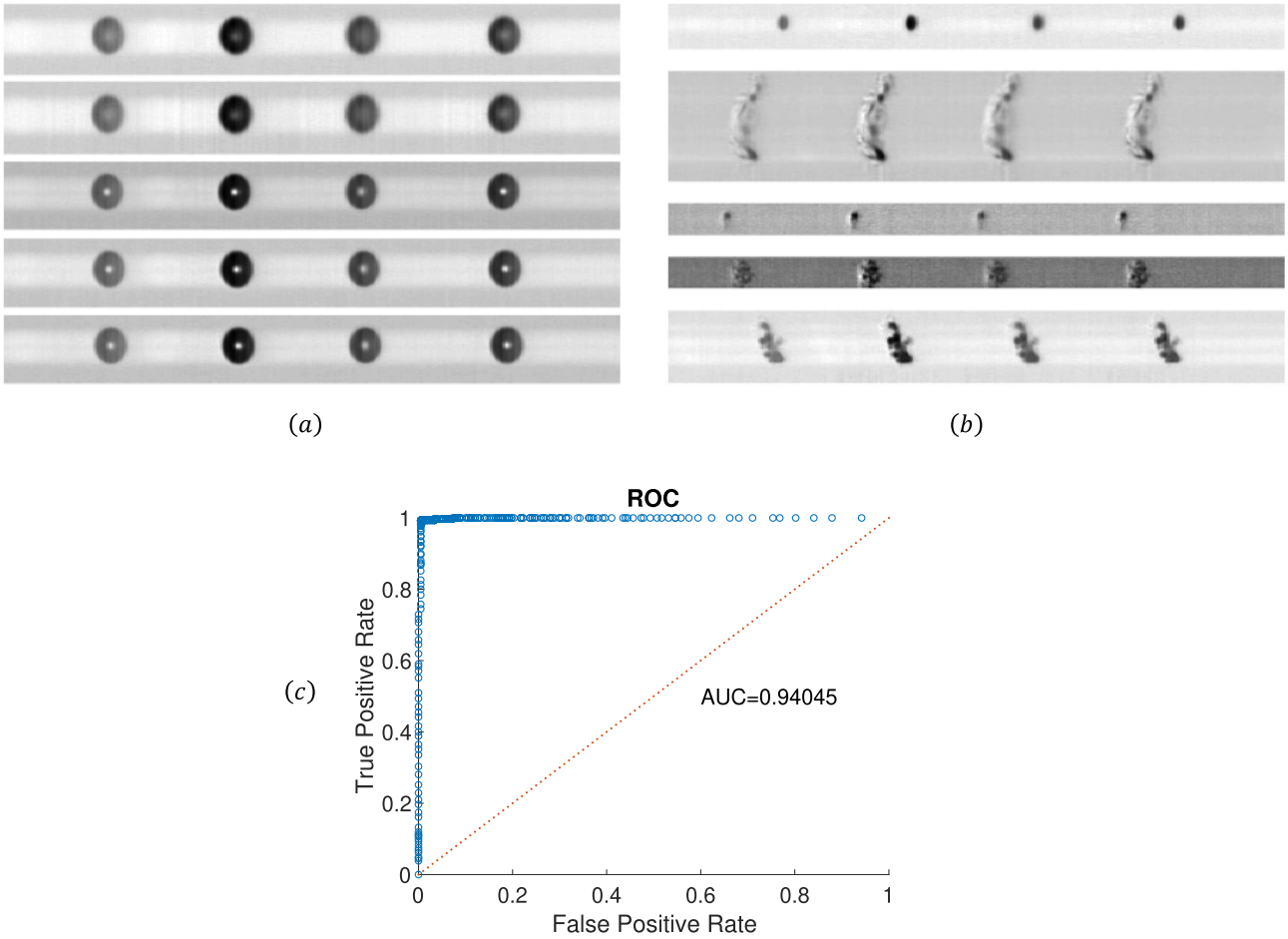


Fig. 6. Object detection using hardware image-based detection and fluorescence detection. (a) Objects detected by both fluorescence and image-based detection modules (true positive examples). (b) Objects identified by image-based module without a corresponding fluorescence signal (false-positive examples). (c) ROC curve showing the image-based detection hardware module is capable of producing comparable detection result as fluorescence detection.

the fluorescence detection signals, a total of 1079 beads were detected in the captured flow. Fig. 6(a) shows samples of beads detected by the image-based detection module, which are also verified by the presence of the fluorescence signal. Fig. 6(b) shows samples of false-positive images detected by the image-based module. These false-positive objects are mostly debris in the flow. As shown in the ROC curve [see Fig. 6(c)], the image-based detection module is capable of

detecting beads with high confidence compared with conventional fluorescence detection methods.

E. Cell Classification Accuracy

We used three subtypes of PBMC, imaged by multi-ATOM, as the target to evaluate our classification system. Each subtype was first negatively isolated (purified) from PBMCs using a targeted magnetic separation protocol with minimal stress and perturbation on the isolated cells. Subsequently, the purified subtypes (each of which was labeled with targeted fluorescence surface marker for validation) were imaged by our system. Flowing and imaging the cells of the purified PBMC subtypes separately allowed us to apply the correct label to the cell images for training, validating, and testing of our QCNN. In total, 4485 B cells, 104 975 monocytes, and 21 114 NK cell images were collected.

To balance the number of each class, 20% of the captured monocyte images were randomly chosen from the whole capture set and combined with all the captured B and NK cells to construct the evaluation data set. The evaluation set was further split into ten shares to perform standard tenfold cross-validation. Overall, our QCNN achieved an average classification top-one accuracy of 0.9545 and an average F1-score of 0.9542 in the tenfold cross-validation.

F. Comparison With GPU-Based Cell Classification

Finally, we have also performed an extensive comparison between our FPGA-based classifier and the state-of-the-art real-time GPU-based cell classifier demonstrated in [11]. To evaluate classification accuracy, the data set released by [11] that contained 65 534 single, aggregated platelet and leukocytes images was used. Since the label for each image was not available, we used the label generated by the pretrained model in [11] as a baseline to evaluate the accuracy of our model. According to the classification results from the pretrained model, there were 18 176 single platelet images, 2205 leukocytes images, and 45 153 aggregated platelet images. Since the number of leukocyte images was too small, they were dropped from our evaluation. Finally, we randomly selected 18 000 images from each of the remaining two cell types to form our evaluation data set. The 36 000 images were split into a 4:1 proportion for training and testing.

The fivefold cross-validation results show that our QCNN was able to achieve classification results that are 94.03% in agreement with the baseline labels produced from [11] while incurring three orders of magnitude lower inference latency. The key factors that result in the significantly reduced classification latency in our system rest on the tight integration between the imaging front end and the QCNN classifier in our case, as well as the use of low-latency hardware QCNN for *in situ* classification. Compared with the system in [11] where image construction and classification were performed using separate computation node connected through 10G Ethernet, our integrated *in situ* classifier eliminates most of the system overheads, including the Ethernet connection and the CPU–GPU communication. Our hardware-based fully pipelined and parallel implementation in combination with cut-through processing of the cell images as they are formed also contributes to the lowered latency. By eliminating most of the variable latency from the network and I/O system, the proposed solution also provides predictable classification latency that facilitates precise control in ultrahigh-speed real-time systems.

VI. Conclusion

We have developed an integrated reconfigurable FPGA-based cell imaging and analytic platform that demonstrates real-time ultralow processing latency needed for applications with real-time feedback requirements. We demonstrate its capability with real-time cell classification using a QCNN, where unmatched low latency is achieved through carefully structured QCNN hardware that allows image formation, cell detection, and computation of different layers of QCNN to execute with pipeline parallelism. The success of our system demonstrates the feasibility and benefits of performing *in situ* image analytics applications using FPGA-based reconfigurable hardware, especially in applications with stringent real-time feedback requirements. Most importantly, the platform is flexible and allows easy reconfiguration, providing a foundation on which systems with similar streaming architecture can be developed for future applications.

References

[1] A. Paszke *et al.*, "Automatic differentiation in PyTorch," in *Proc. NIPS Autodiff Workshop*, 2017, pp. 1–4.

[2] M. Abadi *et al.* (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. [Online]. Available: https://www.tensorflow.org/

[3] A. E. Carpenter *et al.*, "CellProfiler: Image analysis software for identifying and quantifying cell phenotypes," *Genome Biol.*, vol. 7, no. 10, p. R100, Oct. 2006, doi: 10.1186/gb-2006-7-10-r100.

[4] G. Pau, F. Fuchs, O. Sklyar, M. Boutros, and W. Huber, "EBImage—An r package for image processing with applications to cellular phenotypes," *Bioinformatics*, vol. 26, no. 7, pp. 979–981, Apr. 2010, doi: 10.1093/bioinformatics/btq046.

[5] C. Campolo, A. Molinaro, G. Araniti, and A. O. Berthet, "Better platooning control toward autonomous driving: An LTE device-to-device communications strategy that meets ultralow latency requirements," *IEEE Veh. Technol. Mag.*, vol. 12, no. 1, pp. 30–38, Mar. 2017.

[6] M. Khine, C. Ionescu-Zanetti, A. Blatz, L.-P. Wang, and L. P. Lee, "Single-cell electroporation arrays with real-time monitoring and feedback control," *Lab Chip*, vol. 7, no. 4, pp. 457–462, 2007, doi: 10.1039/B614356C.

[7] X. Ding, M. P. Stewart, A. Sharei, J. C. Weaver, R. S. Langer, and K. F. Jensen, "High-throughput nuclear delivery and rapid expression of DNA via mechanical and electrical cell-membrane disruption," *Nature Biomed. Eng.*, vol. 1, no. 3, p. 39, Mar. 2017, doi: 10.1038/s41551-017-0039.

[8] Y.-C. Wu *et al.*, "Massively parallel delivery of large cargo into mammalian cells with light pulses," *Nature Methods*, vol. 12, no. 5, pp. 439–444, Apr. 2015, doi: 10.1038/nmeth.3357.

[9] K. C. M. Lee *et al.*, "Multi-ATOM: Ultrahigh-throughput single-cell quantitative phase imaging with subcellular resolution," *J. Biophotonics*, vol. 12, no. 7, no. 2019, Art. no. e201800479, doi: 10.1002/jbio.201800479.

[10] K. Goda *et al.*, "High-throughput single-microparticle imaging flow analyzer," *Proc. Nat. Acad. Sci. USA*, vol. 109, no. 29, pp. 11630–11635, Jul. 2012. [Online]. Available: https://www.pnas.org/content/109/29/11630

[11] N. Nitta *et al.*, "Intelligent image-activated cell sorting," *Cell*, vol. 175, no. 1, pp. 266–276, Sep. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0092867418310444

[12] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "CNP: An FPGA-based processor for convolutional networks," in *Proc. Int. Conf. Field Program. Log. Appl.*, Aug. 2009, pp. 32–37.

[13] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 1–12.

[14] T. Chen *et al.*, "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *Acm Sigplan Notices*, vol. 49, no. 4, pp. 269–284, 2014.

[15] S. Yin *et al.*, "A high throughput acceleration for hybrid neural networks with efficient resource management on FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 4, pp. 678–691, Apr. 2019.

[16] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Optimizing the convolution operation to accelerate deep neural networks on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 7, pp. 1354–1367, Jul. 2018.

[17] A. Aimar *et al.*, "NullHop: A flexible convolutional neural network accelerator based on sparse representations of feature maps," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 3, pp. 644–656, Mar. 2019.

[18] M. Denil *et al.*, "Predicting parameters in deep learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 2148–2156.

[19] N. Suda *et al.*, "Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2016, pp. 16–25.

[20] Y. Ma, N. Suda, Y. Cao, J.-S. Seo, and S. Vrudhula, "Scalable and modularized RTL compilation of convolutional neural networks onto FPGA," in *Proc. 26th Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2016, pp. 1–8.

[21] M. Motamedi, P. Gysel, and S. Ghiasi, "PLACID: A platform for FPGA-based accelerator creation for DCNNs," *ACM Trans. Multimedia Comput., Commun., Appl.*, vol. 13, no. 4, p. 62, 2017.

[22] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713.

[23] S. Tripathi, G. Dane, B. Kang, V. Bhaskaran, and T. Nguyen, "LCDet: Low-complexity fully-convolutional neural networks for object detection in embedded systems," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jul. 2017, pp. 94–103.

[24] C. Baskin, N. Liss, E. Zheltonozhskii, A. M. Bronstein, and A. Mendelson, "Streaming architecture for large-scale quantized neural networks on an FPGA-based dataflow platform," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2018, pp. 162–169.

[25] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 525–542, doi: 10.1007/978-3-319-46493-0_32.

[26] Y. Umuroglu *et al.*, "FINN: A framework for fast, scalable binarized neural network inference," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 65–74.

[27] R. Zhao *et al.*, "Accelerating binarized convolutional neural networks with software-programmable FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 15–24.

[28] Y. Li, Z. Liu, K. Xu, H. Yu, and F. Ren, "A 7.663-TOPS 8.2-W energy-efficient FPGA accelerator for binary convolutional neural networks (Abstract Only)," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 290–291.

[29] N. J. Fraser *et al.*, "Scaling binarized neural networks on reconfigurable logic," in *Proc. 8th Workshop 6th Workshop Parallel Program. Run-Time Manage. Techn. Many-core Archit. Design Tools Archit. Multicore Embedded Comput. Platforms - PARMA-DITAM*, 2017, pp. 25–30.

[30] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "FP-BNN: Binarized neural network on FPGA," *Neurocomputing*, vol. 275, pp. 1072–1086, Jan. 2018.

[31] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," 2016, *arXiv:1612.01064*. [Online]. Available: http://arxiv.org/abs/1612.01064

[32] A. Prost-Boucle, A. Bourge, F. Petrot, H. Alemdar, N. Caldwell, and V. Leroy, "Scalable high-performance architecture for convolutional ternary neural networks on FPGA," in *Proc. 27th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2017, pp. 1–7.

[33] H. Alemdar, V. Leroy, A. Prost-Boucle, and F. Petrot, "Ternary neural networks for resource-efficient AI applications," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 2547–2554.

[34] S. I. Venieris and C.-S. Bouganis, "Latency-driven design for FPGA-based convolutional neural networks," in *Proc. 27th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2017, pp. 1–8.

[35] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 45–54.

[36] T. Geng *et al.*, "LP-BNN: Ultra-low-latency BNN inference with layer parallelism," in *Proc. IEEE 30th Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, Jul. 2019, pp. 9–16.

[37] Y. Park, C. Depeursinge, and G. Popescu, "Quantitative phase imaging in biomedicine," *Nature Photon.*, vol. 12, no. 10, pp. 578–589, Oct. 2018, doi: 10.1038/s41566-018-0253-x.

[38] K. C. M. Lee *et al.*, "Quantitative phase imaging flow cytometry for ultra-large-scale single-cell biophysical phenotyping," *Cytometry A*, vol. 95, no. 5, pp. 510–520, May 2019, doi: 10.1002/cyto.a.23765.

[39] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[40] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, 2017.

[41] J. Hickish *et al.*, "A decade of developing radio-astronomy instrumentation using CASPER open-source technology," *J. Astronomical Instrum.*, vol. 5, no. 4, Dec. 2016, Art. no. 1641001, doi: 10.1142/S2251171716410014.

Maolin Wang received the B.Eng. degree in electrical engineering from Tsinghua University, Beijing, China, in 2015, and the Ph.D. degree from the Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong, in 2020.

He is currently a Post-Doctoral Fellow with The University of Hong Kong. His research interests include efficient hardware architectures and algorithms for the training and inference of deep neural networks.

Kelvin C. M. Lee received the B.Eng. degree in medical engineering and the Ph.D. degree in electrical and electronic engineering from The University of Hong Kong, Hong Kong, in 2015 and 2019, respectively.

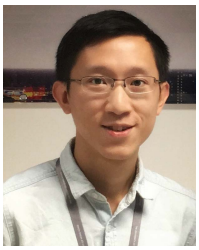
He is currently a Post-Doctoral Fellow in the Department of Electrical and Electronic Engineering, The University of Hong Kong. His research interests cover high-throughput imaging flow cytometry, inertial microfluidics, and biophotonics for translational medicine.

Bob M. F. Chung received the B.Eng. degree in medical engineering and the Ph.D. degree in mechanical engineering from The University of Hong Kong, Hong Kong, in 2013 and 2020, respectively.

He is currently a Post-Doctoral Fellow with the Department of Electrical and Electronic Engineering, The University of Hong Kong. His research focus is on the microfluidic cell sorting in high-throughput single-cell imaging platform and its implementation in biomedical applications.

Sharatchandra Varma Bogaraju received the master's degree in VLSI-CAD from Manipal University, Manipal, India, and the Ph.D. degree from IIT Delhi, New Delhi, India, in 2007 and 2015, respectively.

He was a Post-Doctoral Research Fellow with Queen's University Belfast, Belfast, U.K., and The University of Hong Kong, Hong Kong. He was an Assistant Professor with the National Institute of Technology Goa, Ponda, India, and a Visiting Faculty with the Indian Institute of Space Science and Technology (IIST) Thiruvananthapuram, Thiruvananthapuram, India. He is currently a Lecturer with the Faculty of Computing, Engineering and the Built Environment, Ulster University, Jordanstown Campus, U.K. His research interests include field-programmable gate array (FPGA)-based hardware accelerators, reconfigurable computing, and bioinformatics.

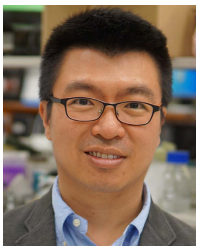


Ho-Cheung Ng received the B.Eng. and M.Phil. degrees from The University of Hong Kong, Hong Kong, in 2012 and 2016, respectively. He is currently pursuing the Ph.D. degree with the Department of Computing, Imperial College London, London, U.K.

His research interests include high-performance computing, real-time image processing, and computational biology.

Justin S. J. Wong received the M.Eng. and Ph.D. degrees in electrical and electronic engineering from Imperial College London, London, U.K., in 2006 and 2011, respectively.

He was a Research Associate with the Circuits and Systems Group, Imperial College London, in 2014, and received the Charted Engineer (CEng) Qualification. He then worked in ZMP Inc., Tokyo, Japan, on real-time sensor and imaging systems for autonomous vehicles until 2016. He is currently the Vice President Chief Engineer of Conzeb Ltd., Hong Kong, and is collaborating closely with HKU to develop field-programmable gate array (FPGA)-based ultrahigh throughput real-time imaging and classification systems for a cancer diagnostic. His research interests include ultrahigh-speed real-time image processing, super-resolution, and convolutional neural network (CNN)-based image classification on FPGAs and graphic processing units (GPUs).



Ho Cheung Shum received the B.S.E. degree *(summa cum laude)* in chemical engineering from Princeton University, Princeton, NJ, USA, in 2005, and the M.S. and Ph.D. degrees in applied physics from Harvard University, Cambridge, MA, USA, in 2007 and 2010, respectively.

He is currently a Professor with the Department of Mechanical Engineering, The University of Hong Kong, Hong Kong. His research interests include biomicrofluidics and soft matter.

Dr. Shum is a fellow of the Royal Society of Chemistry, a Founding Member of the Hong Kong Young Academy of Sciences, and an Associate Editor for *Biomicrofluidics* [American Institute of Physics (AIP)].

Kevin K. Tsia (Member, IEEE) received the Ph.D. degree from the Electrical Engineering Department, University of California at Los Angeles (UCLA), Los Angeles, CA, USA, in 2009.

He is currently a Professor with the Department of Electrical and Electronic Engineering and the Program Director of the Biomedical Engineering Program, The University of Hong Kong, Hong Kong. He holds four granted and four pending U.S. patents on ultrafast optical imaging technologies. His research interest covers ultrafast optical imaging for imaging flow cytometry and cell-based assay, high-speed *in vivo* brain imaging, and single-cell analysis.

Dr. Tsia is the HK Research Grants Council (RGC) Research Fellow.

Hayden Kwok-Hay So (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer sciences from the University of California at Berkeley, Berkeley, CA, USA, in 1998, 2000, and 2007, respectively.

He is currently an Associate Professor and the Co-Director of the Computer Engineering Program at the Department of Electrical and Electronic Engineering and the Co-Director of the Joint Lab on Future Cities, The University of Hong Kong, Hong Kong. His research focuses on highly efficient reconfigurable computing systems and their applications.

Dr. So received the Croucher Innovation Award in 2013, the University Outstanding Teaching Award (Team) in 2012, and the Faculty Best Teacher Award in 2011. He was the Technical Program Chair for various international conferences, including the ARC 2020, ASAP 2015, FPT 2014, and HEART 2014. He was also the Multiprocessor Systems and Networks on Chip Track Co-Chair of ReConfig and a Guest Editor of the *Journal of Signal Processing Systems*.